UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 10/684,053 | 10/09/2003 | Chandan Mathur | 1934-12-3 | 3240 |

7590 12/27/2010

Bryan A. Santarelli
GRAYBEAL JACKSON HALEY LLP
Suite 350
155-108th Avenue NE
Bellevue, WA 98004-5901

| EXAMINER | |
|---|---|
| HUISMAN, DAVID J | |

| ART UNIT | PAPER NUMBER |
|---|---|
| 2183 | |

| MAIL DATE | DELIVERY MODE |
|---|---|
| 12/27/2010 | PAPER |

**Please find below and/or attached an Office communication concerning this application or proceeding.**

The time period for reply, if any, is set in the attached communication.

| Office Action Summary | Application No. | Applicant(s) | |
|---|---|---|---|
| | 10/684,053 | MATHUR ET AL. | |
| | Examiner | Art Unit | |
| | DAVID J. HUISMAN | 2183 | |

*-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --*

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

1)☒ Responsive to communication(s) filed on *08 October 2010*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

4)☒ Claim(s) *1-15 and 17-62* is/are pending in the application.

4a) Of the above claim(s) *25-36 and 55-61* is/are withdrawn from consideration.

5)☒ Claim(s) *19-24,51-54 and 62* is/are allowed.

6)☒ Claim(s) *1-15,17,18 and 37-50* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

**Application Papers**

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *14 May 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *6/1/2010 & 10/8/2010*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

## DETAILED ACTION

1. Claims 1-15 and 17-62 are pending. Claims 25-36 and 55-61 have been withdrawn. Claims 1-15, 17-24, 37-54, and 62 have been examined.

### Information Disclosure Statement

2. In the IDS filed on October 8, 2010, NPL document 9 has not been considered (denoted by strike-through) because 37 CFR 1.98(a)(2)(ii) requires that a legible copy be provided. However, the copy provided is very difficult to read and some parts are completely illegible.

### Claim Objections

3. Claim 1 is objected to because of the following informalities: In line 4, replace the comma after "to" with a colon. Appropriate correction is required.

4. Claim 42 is objected to because of the following informalities: In line 2, replace "comprise" with --comprises--. Appropriate correction is required.

5. Claim 62 is objected to because of the following informalities: In line 4, it appears that "to", at the end of the line, is followed by a semicolon instead of a colon. Appropriate correction is required. Also, please check other claims for this mistake.

### Claim Rejections - 35 USC § 102

6. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

7. Claims 1-4, 6, 8, 10-13, 15, 18, 37-42, and 45-49 are rejected under 35 U.S.C. 102(e) as being anticipated by Bass et al., U.S. Patent No. 6,405,266 (herein referred to as Bass).

8. Referring to claim 1, Bass has taught a computing machine, comprising:

a) first and second parallel buffers respectively associated with first and second data processing units. See Fig.1 and column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a first buffer would exist between objects 1 and 2 and a second buffer would exist between objects 1 and 4.

b) a processor coupled to the buffers (this is deemed inherent, as data is sent from buffers to processor logic for processing) and operable to:

b1) execute an application and first, second, third, and fourth data-transfer objects. All processors inherently execute an application. Also, first through fourth objects are executed as described below.

b2) publish data under the control of the application. An application produces data that is to be sent to a subscribing object by way of a publishing object. For instance, see column 6, lines 1-5. An application publishes a message via a publishing object.

b3) load at least a portion of the published data into the first buffer under the control of the first data-transfer object. See Fig.1 and column 6, lines 1-5. Object 1 (publishing object) will load data into the buffer between it and object 2, for instance, which is a particular subscribing object.

b4) load at least the same portion of the published data into the second buffer under the control of the second data-transfer object. See column 5, lines 52-55. The same data is automatically outputted to the external broker object 103 by message broker object 108 (i.e., the second data-transfer object). Hence, the second data transfer object plays at least some part in loading the same data into the buffer between object 1 and object 4, for instance, which is another particular subscribing object.

b5) retrieve at least the portion of the published data from the first and second buffers under the control of the third and fourth data-transfer object, respectively. See Fig.1, column 3, line 66, to column 4, line 4, and column 6, lines 1-5. The third and fourth objects (i.e., subscribing objects 2 and 4, respectively, retrieve the data asynchronously from the respective buffers). Note also, that one of the third and fourth objects may comprise broker object 113.

9. Referring to claim 2, Bass has taught the computing machine of claim 1 wherein:

a) the first and third data-transfer objects respectively comprise first and second instances of first object code. Note that, in general, the first and third objects comprise message communicating code.

b) the second and fourth data-transfer objects respectively comprise first and second instances of second object code. Similar to above, the second and fourth objects comprise message communicating code.

10. Referring to claim 3, Bass has taught the computing machine of claim 1 wherein the processor comprises:

a) a processing unit operable to execute the application and publish the data under the control of the application. Recall from the rejection of claim 1 that the processor inherently executes an application and publishes data under control of the application. This execution and publishing is performed by a processing unit.

b) a data-transfer handler operable to execute the first, second, third, and fourth data-transfer objects, to load the published data into the first and second buffers under the control of the first and second data-transfer objects, respectively, and to retrieve the published data from the first and second buffers under the control of the third and fourth data-transfer objects, respectively. Again, recall from the rejection of claim 1 that first and second objects loading the first and second buffers, respectively, and third and fourth objects retrieve the data from first and second buffers, respectively. The unit which performs this execution for loading and retrieving data is a data transfer handler.

11. Referring to claim 4, Bass has taught the computing machine of claim 1 wherein the processor is further operable to execute a thread of the application and to publish the data under the control of the thread. See column 3, line 66, to column 4, line 4, and column 5, lines 30-32. Note that the system executes threads.

12. Referring to claim 6, Bass has taught the computing machine of claim 1, further comprising:

a) a bus. This is deemed inherent in a communication system (to carry communications from one location to another).

b) wherein the processor is operable to execute a communication object and to drive the data retrieved from one of the first and second buffers onto the bus under the control of the communication object. See Fig.1, at least component 116, which controls communication.

13. Referring to claim 8, Bass has taught the computing machine of claim 1 wherein the processor is further operable to generate a message that includes a header and data retrieved from one of the first and second buffers under the control of the respective one of the third and fourth data-transfer objects. It should be noted that, as messages are passed between objects 108, 103, and 113, destinations of the messages must be generated (so that it is known which broker to send the message to). Hence, the destination can be considered the header. In addition, even if this were somehow proven untrue (which the examiner does not believe is possible), generating headers for messages is known in the art. Headers provide useful information about the message being transmitted and are used to assist in the communication process. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that the processor is further operable to generate a message that includes a header and data retrieved from one of the first and second buffers under the control of the respective one of the third and fourth data-transfer objects. Note, from above, however, that this claim is anticipated.

14. Referring to claim 10, Bass has taught a computing machine, comprising:

a) a first buffer. See column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a first buffer would exist between objects 1 and 2. Note also that object 1 may have objects 3 and 4 as

subscribers. Though, this is not explicitly disclosed, one of ordinary skill would realize that any object could subscribe to any particular published item.

b) a processor coupled to the buffer (this is deemed inherent, as data is sent from the buffer to processor logic for processing) and operable to:

b1) execute first and second data-transfer objects and an application.

b2) generate data under control of the application such that the generated data includes no data-destination information. The application inherently generates data. And, as is known with the pub/sub protocol, no data destination information is generated. See column 1, lines 30-34, and column 3, lines 55-58.

b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer object. See column 6, lines 1-5. Object 1 gets the data from the application and loads it into a buffer for asynchronous communication.

b4) unload the data from the buffer under the control of the second data-transfer object. See Fig.1, object 2. This second data transfer object, by subscribing to published data of object 1, will read that data from the buffer. Or, the second data transfer object could be object 103, which unloads that data from the first buffer into another buffer for communication with process B (object 4).

b5) process the unloaded data under the control of the application such that the processed data includes only non-data-destination information. Clearly, message data is processed. And, recall that no data destination information is present.

15. Referring to claim 11, Bass has taught the computing machine of claim 10 wherein the first and second data-transfer objects respectively comprise first and second instances of the same object code. Note that, in general, the first and second objects comprise message communicating code. Hence, they are instances of the same object code.

16. Referring to claim 12, Bass has taught the computing machine of claim 10 wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

17. Referring to claim 13, Bass has taught the computing machine of claim 10 wherein the processor is further operable to execute first and second threads of the application, generate the data under the control of the first thread, and process the unloaded data under the control of the second thread. See column 3, line 66, to column 4, line 4, and column 5, lines 30-32. Note that the system executes threads (e.g., process/thread A and process/thread B in Fig.1). When object 1 communicates with object 4, the first thread generates the data, and the second thread processes the unloaded/sent data.

18. Referring to claim 15, Bass has taught the computing machine of claim 10, further comprising:

a) a second buffer. See column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a second buffer would exist between objects 1 and 4.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. Again, the second object, under one interpretation (when object 2 is not a subscriber), would be object 103, which would unload data from the first buffer in object 108 and store it in a second buffer, from which object 3 or object 113, for instance, would retrieve it for processing by the application. Object 3 or object 113 may be considered the third data transfer object.

19. Referring to claim 18, Bass has taught the computing machine of claim 10 wherein the processor is further operable to package the generated data into a message that includes a header and the data under the control of the second data-transfer object. It should be noted that, as messages are passed between objects 108, 103, and 113, destinations of the messages must be generated (so that it is known which broker to send the message to). Hence, the destination can be considered the header. In addition, even if this were somehow proven untrue (which the examiner does not believe is possible), generating headers for messages is known in the art. Headers provide useful information about the message being transmitted and are used to assist in

the communication process. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that package the generated data into a message that includes a header and the data under the control of the second data-transfer object. Note, from above, however, that this claim is anticipated. Also, note that by simply forwarding the message on (even if it was previously packaged), all of the message data/header is grouped and sent on. So, it is, in a sense, repackaged.

20. Referring to claim 37, Bass has taught a method comprising:

a) publishing data with an application that does not generate an address of a destination of the data. See Fig.1. An application publishes data by way of publishing objects. This is also done without generating a destination address, as is known in the publishing/subscribing protocol. See column 1, lines 30-34, and column 3, lines 55-58.

b) loading the published data into a first buffer with a first data-transfer object, the loaded data absent the address of the destination of the data, each location within the buffer corresponding to the address. See Fig.1 and column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a first buffer would exist between objects 1 and 2 or 1 and 4 (or even 1 and 3 if object 3 subscribed to the data published by object 1). Specifically, for published data to get from object 1 to a subscribing object, at least one message broker must be used. Therefore, the buffer is part of the broker. Again, note that an address is not stored in this buffer because the publishing object does not know the destination.

c) retrieving the published data from the buffer with a second data-transfer object, the retrieved data including no information indicating a destination of the data. See Fig.1, column 1, lines 30-34, and column 6, lines 1-5. Note that another object will remove the data from the buffer in order to transmit it to the destination. Again, no address is included in the data.

d) generating a message header that includes a destination of the retrieved data. See Fig.1 and note that the brokers inherently generate addresses since they may send data to multiple locations via publish commands. This address is part of a header.

e) generating a message that includes the retrieved data and the message header. See Fig.1. Broker 103, for instance, sends a message to either broker 108 or 113, or both. Therefore, a destination must be provided with the message.

21. Referring to claim 38, Bass has taught the method of claim 37, wherein publishing the data comprises publishing the data with a thread of the application. See column 3, line 66, to column 4, line 4, and column 5, lines 30-32. Note that the system executes threads.

22. Referring to claim 39, Bass has taught the method of claim 37, further comprising: generating a queue value that corresponds to the presence of the published data in the buffer, notifying the second data-transfer object that the published data occupies the buffer in response to the queue value, wherein retrieving the published data comprises retrieving the published data from the buffer with the second data-transfer object in response to the notification. This is deemed inherent. Clearly, some signal/value must be generated which indicates data is received in a buffer and that it must be retrieved.

23. Referring to claim 40, Bass has taught the method of claim 37, further comprising driving the message onto a bus with a communication object. See Fig.1, at least component 116, which controls communication.

24. Referring to claim 41, Bass has taught the method of claim 37, further comprising loading the retrieved data into a second buffer with the second data-transfer object. Each broker has at least one buffer to store message sent to it. Therefore, second data-transfer object 108 would load data from the first buffer into a buffer in broker 103. Or, second data transfer object 103 would load data from the first buffer into a buffer in broker 113.

25. Referring to claim 42, Bass has taught the method of claim 37 wherein generating the message header and the message comprise generating the message header and the message with the second data transfer object. See Fig.1 and note the messages with headers are generated when they are passed between brokers. The initial sources and ultimate destinations know nothing about the addresses, as is known in the pub/sub protocol.

26. Referring to claim 45, Bass has taught a method comprising:

a) receiving a message that includes data and that includes a message header that indicates a destination address of the data, the destination address corresponding to a software application. See Fig.1. Message are passed between processes/applications on the same or different processors (column 6, lines 40-49). As discussed throughout Bass, and in column 1, lines 21-33, process objects communicate by way of a pub/sub protocol through a series of brokers. As is known, the initial object that publishes the data has no knowledge of the ultimate destination address. Similarly, the ultimate destination has no idea of the sender's address. However, it is inherent, as messages are passed between brokers, that addresses in headers accompany them.

For instance, broker 103 can send to one or more processes/applications, and therefore, must know which address to send a message to. Hence, broker 103 will retrieve a message from one process and attach an address of another process broker 108/113.

b) loading into a first buffer with a first data-transfer object, the received data without the message header, the first buffer corresponding to the destination. See Fig.1 and column 2, lines 18-25. Multiple objects may subscribe to published data. As disclosed, queues/buffers exist between objects for asynchronous communication. For example, in column 6, lines 1-5, object 1 sends data to objects 2 and 4. Hence, a first buffer would exist to hold the message sent from broker 103 to object 4. Note that the data and the header are separate so, the data is stored without the header. Also, it is known that a destination is stripped when it arrives to the destination. Storing it is generally unnecessary and would require additional memory to do so.

c) unloading the data from the buffer with a second data-transfer object and processing the unloaded data with an application corresponding to the destination.. See Fig.1 and column 6, lines 1-5. A second object (either broker 113 or destination object 4) will unload the data from the buffer for processing by the application process.

27. Referring to claim 46, Bass has taught the method of claim 45 wherein processing the unloaded data comprises processing the unloaded data with a thread of the application corresponding to the destination. See column 3, line 66, to column 4, line 4, and column 5, lines 30-32. Note that the system executes threads.

28. Referring to claim 47, Bass has taught the method of claim 45, further comprising: generating a queue value that corresponds to the presence of the data in the buffer, notifying the second data-transfer object that the published data occupies the buffer in response to the queue

value, wherein unloading the data comprises unloading the data from the buffer with the first data-transfer object in response to the notification. This is deemed inherent. Clearly, some signal/value must be generated which indicates data is received in a buffer and that it must be retrieved/unloaded. Broker 113 would then send the data to broker 113.

29. Referring to claim 48, Bass has taught the method of claim 45, further comprising wherein receiving the message comprises receiving the message with the first data-transfer object. See Fig.1. Broker 103 receives the data.

30. Referring to claim 49, Bass has taught the method of claim 45, further comprising: receiving the message comprises retrieving the message from a bus with a communication object and transferring the data from the communication object to the first data transfer object. See Fig.1. Communication object 108 or 116 retrieves the data from the initial source and sends it on to broker 103.

**Claim Rejections - 35 USC § 103**

31. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

32. Claim 7 is rejected under 35 U.S.C. 103(a) as being unpatentable over Bass.

33. Referring to claim 7, Bass has taught the computing machine of claim 1. Bass has not taught a third buffer. However, one of ordinary skill in the art would have recognized that there may be more than just two subscribing objects. Pub/sub allows for scalability and for more

subscribers to exist if desired. Hence, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass to include another object (either in process B or in another process (e.g., process C) which subscribes to publishings by object 1 via a third buffer. Then, Bass, as modified, has taught that the processor is operable to provide the data retrieved from one of the first and second buffers to the third buffer under the control of the respective one of the third and fourth data-transfer objects. See Fig.1. If the new object is part of process B then third/fourth object 113 would transfer the data from the first buffer managed by process A into the third buffer managed by process B.

34. Claims 5, 9, 14, 17, 43-44, and 50 are rejected under 35 U.S.C. 103(a) as being unpatentable over Bass in view of the examiner's taking of Official Notice.

35. Referring to claim 5, Bass has taught the computing machine of claim 1. Bass has further taught that the processor is further operable to:

a) store a queue value, the queue value reflecting the loading of the published data into the first buffer, read the queue value, notify the third data-transfer object that the published data occupies the first buffer in response to the queue value, and retrieve the published data from the first buffer under the control of the third data-transfer object and in response to the notification. This is deemed inherent. Clearly when data is received, some value must be generated and read to indicate and notify that data is available in the queue for transmission/retrieval.

b) Bass has not taught executing a queue object and a reader object, the queue value is stored under the control of the queue object, the queue value is read under the control of the reader object, and the third data-transfer object is notified that the published data occupies the first

buffer under the control of the reader object. However, since Bass's environment is an object-oriented environment, one of ordinary skill in the art would have recognized that components may be programmed as software objects, including the components which generate and read the queue values and/or perform notification. Since software may be quickly modified to change functionality, design of these components is more flexible compared to dedicated hardware components for performing the claimed functions. If a problems exists in the hardware, it must be rebuilt instead of reprogrammed, which is more difficult and time-consuming. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass to execute a queue object and a reader object, to store the queue value under the control of the queue object, to read the queue value under the control of the reader object, and to notify the third data-transfer object that the published data occupies the first buffer under the control of the reader object.

36. Referring to claim 9, Bass has taught the computing machine of claim 1 wherein:

a) the first and third data-transfer objects respectively comprise first and second instances of first object code. Note that, in general, the first and third objects comprise message communicating code.

b) the second and fourth data-transfer objects respectively comprise first and second instances of second object code. Similar to above, the second and fourth objects comprise message communicating code.

c) Bass has not taught that the processor is operable to execute an object factory and to generate the first object code and the second object code under the control of the object factory. However, object factories and their advantages are known in the art of object oriented

programming. Specifically, an object factory is an object used to create other objects, and has proven advantages. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that the processor is operable to execute an object factory and to generate the first object code and the second object code under the control of the object factory.

37. Referring to claim 14, Bass has taught the computing machine of claim 10. Bass has further taught that the processor is further operable to:

a) store a queue value, the queue value reflecting the loading of the retrieved data into the first buffer, read the queue value, notify the second data-transfer object that the retrieved data occupies the buffer in response to the queue value, and unload the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. This is deemed inherent. Clearly when data is received, some value must be generated and read to indicate and notify that data is available in the queue for transmission/retrieval.

b) Bass has not taught executing a queue object and a reader object, the queue value is stored under the control of the queue object, the queue value is read under the control of the reader object, and the second data-transfer object is notified that the retrieved data occupies the buffer under the control of the reader object. However, since Bass's environment is an object-oriented environment, one of ordinary skill in the art would have recognized that components may be programmed as software objects, including the components which generate and read the queue values and/or perform notification. Since software may be quickly modified to change functionality, design of these components is more flexible compared to dedicated hardware components for performing the claimed functions. If a problems exists in the hardware, it must

be rebuilt instead of reprogrammed, which is more difficult and time-consuming. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass to execute a queue object and a reader object, to store the queue value under the control of the queue object, to read the queue value under the control of the reader object, and to notify the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object.

38. Referring to claim 17, Bass has taught the computing machine of claim 10 wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. Note that, in general, the first and second objects comprise message communicating code. Hence, they are instances of the same object code.

b) Bass has not taught that the processor is operable to execute an object factory and to generate the object code under the control of the object factory. However, object factories and their advantages are known in the art of object oriented programming. Specifically, an object factory is an object used to create other objects, and has proven advantages. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that the processor is operable to execute an object factory and to generate the object code under the control of the object factory.

39. Referring to claim 43, Bass has taught the method of claim 37, further comprising:

a) generating the first data-transfer object as a first instance of data-transfer object code and generating the second data-transfer object as a second instance of the object code. Note that, in general, the first and second objects comprise message communicating code. Therefore, they are instances of communication object code.

b) Bass has not taught generating the object code with an object factory. However, object factories and their advantages are known in the art of object oriented programming. Specifically, an object factory is an object used to create other objects, and has proven advantages. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass such that the processor is operable to execute an object factory and to generate the object code under the control of the object factory.

40. Referring to claim 44, Bass has taught the method of claim 37. Bass has not taught receiving the message and processing the data in the message with a hardwired pipeline accelerator. However, Official Notice is taken that hardwired pipelined processors and their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass's receiving processor such that it includes a hardwired pipeline for accelerating execution.

41. Referring to claim 50, Bass has taught the method of claim 45. Bass has not explicitly taught generating the message header and the message with a hardwired pipeline accelerator. However, Official Notice is taken that hardwired pipelined processors and their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Bass's processor executing broker code 103 such that it includes a hardwired pipeline for accelerating execution.

42. Claims 10-15, 17-18, and 45-50 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dretzka et al., U.S. Patent No. 4,703,475 (herein referred to as Dretzka), in view of the examiner's taking of Official Notice. Note that some of the claims are rejected multiple times under different interpretations of Dretzka.

43. Referring to claim 10, Dretzka has taught a computing machine, comprising:

a) a first buffer. See Fig.6, buffer 220-4, for instance.

b) a processor (Fig.1, component 21) coupled to the buffer and operable to:

b1) execute first and second data-transfers and an application. Fig.4 sets forth at least some of the data-transfers executed by processor 21. Looking at Fig.4 and Fig.6, a first transfer would be executed to load data into buffer 220-4 and a second transfer would be executed to unload data from buffer 220-4. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.

b2) generate data under control of the application without generating an address of a destination of the data. See Fig.6 and Figs.8-15, and note that the level 2.5 application generates data using an input list by searching for consecutive sequence packets. Also, note from Fig.4, at level 2.5, the 1-byte header (data-destination information) is deleted, and therefore, the data includes no data-destination information. In an alternate interpretation, if an address is inherently generated in Dretzka, as applicant argues, the address may still be considered as being separate from the generated data (i.e., the true

content of the message). Therefore, the generated data itself does not include any data-destination information and, further, the data is generated without generating the address.

b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer. See Fig.6. After data is generated by the input list, the first transfer transfers it to the buffer (i.e., 220-4).

b3) unload the data from the buffer under the control of the second data-transfer. See Fig.6. Data is ultimately moved/unloaded from the buffer 220-4 and into buffer 210 under control of the second transfer.

b4) process the unloaded data under the control of the application without receiving the address of the destination of the data. See Fig.4 and Fig.6. From the buffer 210, the processor will process the data using an application. Also, note from Fig.4, at level 2.5 (which occurs well before the unloading), the 1-byte header (data-destination information) is deleted, and therefore, the processed data includes no data-destination information. In the alternate interpretation, if an address inherently exists, as argued by applicant, then this address is still separate from the generated data. Hence, data is processed separately from the address. That is the address does not need to be received to process the data.

b5) Dretzka has not taught executing first and second data-transfer objects. However, object-oriented programming and objects are known and have known advantages in the art. Therefore, particular features of Dretzka may be implemented as objects. For instance, a loader for loading data into a buffer may be implemented as an object. Likewise, an unloader for unloading data from a buffer may be implemented as an object.

Since object-oriented programming has known benefits to at least some programmers, such as making programs easier to manage and keep track of, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor executes first and second data transfer objects to carry out the respective transfers.

44. Referring to claim 11, Dretzka, as modified, has taught the computing machine of claim 10 wherein the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.6 and note that, in general, the first object retrieves data from a previous stage (level 2.5 stage) and stores it in a next buffer 220-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 210. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

45. Referring to claim 12, Dretzka, as modified, has taught the computing machine of claim 10 wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 (and from Fig.4) that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

46. Referring to claim 13, Dretzka, as modified, has taught the computing machine of claim 10. Dretzka has not taught that the processor is further operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, since threads are independent sequences of instructions, the system may switch to a next thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, and because generating data and processing unloaded data are independent tasks, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor is operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread.

47. Referring to claim 14, Dretzka, as modified, has taught the computing machine of claim 10 wherein the processor is further operable to:

a) execute a queue object and a reader object. See Fig.4 and Fig.6. The queue object is the object which stores the more bit, which ultimately leads to the informing of level 4 that a complete message has been received. The reader object is the object which detects the more bit so that further action can occur.

b) store a queue value under the control of the queue object, the queue value reflecting the loading of the retrieved data into the first buffer. See Fig.4 and Fig.6. The queue object will

store the "more" bit, which reflects the loading of the retrieved data into the buffer. This bit, when set in a certain manner, will allow for the informing of level 4 of a complete message.

c) read the queue value under the control of the reader object. See Fig.4 and note that the more bit, when set to a certain level, will indicate the end of the message and that the message may be further transmitted and processed. The reader object will have to read this more bit.

d) notify the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object and in response to the queue value. When the "more" bit is set in the appropriate manner and noted by the reader object, the data may be transferred to the next-level buffer. See Fig.4 and Fig.6.

e) unload the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. Again, in response to the notification, the data will be moved from level 3 buffer to level 4 buffer. See Fig.4 and Fig.6.

f) Note that while queue and reader objects are not explicitly recited in Dretzka, implementing the queue/reader functionality as objects for execution is obvious for the reasons set forth in claim 10.

48. Referring to claim 15, Dretzka, as modified, has taught the computing machine of claim 10, further comprising:

a) a second buffer. See Fig.6, component 210.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. See Fig.4 and Fig.6. Data is unloaded from buffer 220-4 to buffer 210

under control of the second transfer object, and then moved from buffer 210 to the application in the processor under control of the third object.

c) Note that while the third object is not explicitly recited in Dretzka, implementing the third data transfer via object execution is obvious for the reasons set forth in claim 10.

49. Referring to claim 17, Dretzka, as modified, has taught the computing machine of claim 10 wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.6 and note that, in general, the first object retrieves data from a previous stage (level 2.5 stage) and stores it in a next buffer 220-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 210. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

b) Dretzka has not taught that the processor is operable to execute an object factory and to generate the object code under the control of the object factory. However, object factories and their advantages are known in the art of object oriented programming. Specifically, an object factory is an object used to create other objects, and has proven advantages. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor is operable to execute an object factory and to generate the object code under the control of the object factory.

50. Referring to claim 10, Dretzka has taught a computing machine (under a second interpretation), comprising:

a) a first buffer. See Fig.5, buffer 110, for instance.

b) a processor (Fig.1, component 11) coupled to the buffer and operable to:

b1) execute first and second data-transfers and an application. Fig.3 sets forth at least some of the data-transfers executed by processor 11. Looking at Fig.3 and Fig.5, a first transfer would be executed to load data into buffer 110 and a second transfer would be executed to unload data from buffer 110 and into buffer 120-4, for instance. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.

b2) generate data under control of the application without generating an address of a destination of the data. See Fig.5 and note that a message comprising data must inherently be generated before being stored in buffer 110. This data, as shown at the top of Fig.3, is generated as a result of application execution. Note that at this time of generation, no headers and/or data-destination information are attached to the generated data. In an alternate interpretation, if an address is inherently generated in Dretzka, as applicant argues, the address is separate from generated data. Therefore, the generated data itself does not include any data-destination information and, further, the data is generated without generating the address.

b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer. See Fig.5. After data is generated, it is ultimately stored in buffer 110.

b3) unload the data from the buffer under the control of the second data-transfer. See Fig.5. Data is ultimately moved/unloaded from the buffer 110 and into buffer 120-4 under control of the second transfer.

b4) process the unloaded data under the control of the application without receiving the address of the destination of the data. See Fig.3 and note that the data, after being moved into buffer 120-4, is further processed by breaking the message up. It isn't until the data is in the next buffer that a 1-byte channel header, which may or may not be considered data-destination information is added to it. Hence, at the time of the claimed processing, since the 1-byte header has not been added, the data does not include data-destination information. In the alternate interpretation, if an address inherently exists, as argued by applicant, then this address is still separate from the generated data. Hence, data is processed separately from the address. That is the address does not need to be received to process the data.

b5) Dretzka has not taught executing first and second data-transfer objects. However, object-oriented programming and objects are known and have known advantages in the art. Therefore, particular features of Dretzka may be implemented as objects. For instance, a loader for loading data into a buffer may be implemented as an object. Likewise, an unloader for unloading data from a buffer may be implemented as an object. Since object-oriented programming has known benefits to at least some programmers, such as making programs easier to manage and keep track of, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor executes first and second data transfer objects to carry out the respective transfers.

51. Referring to claim 11, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation) wherein the first and second data-transfer objects respectively

comprise first and second instances of the same object code. See Fig.5 and note that, in general, the first object retrieves data from a source and stores it in a next buffer 110. Likewise, the second object retrieves data from a source (buffer 110) and stores it in a next buffer 120-4. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

52. Referring to claim 12, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation) wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 (and from Fig.3) that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

53. Referring to claim 13, Dretzka, as modified, has taught the computing machine of claim 10. Dretzka has not taught that the processor is further operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, since threads are independent sequences of

instructions, the system may switch to a next thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, and because generating data and processing unloaded data are independent tasks, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor is operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread.

54. Referring to claim 14, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation).

a) Dretzka has not taught that the processor is further operable to execute a queue object and a reader object. However, recall from Fig.5 that data is initially stored in a queue. The examiner asserts that it is well known and advantageous to have an empty bit indicate the status of the queue because it prevents the queue from being read if it has no useful data in it, thereby saving time. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka to execute a queue object to set/clear an empty bit based on whether the queue contains valid data to be read, and also a reader object to read the empty bit such that the system knows when to read the valid data from the queue.

b) Dretzka, as modified, has further taught storing a queue value under the control of the queue object, the queue value reflecting the loading of the retrieved data into the first buffer. This is deemed inherent as an empty bit would be stored.

c) Dretzka, as modified, has further taught reading the queue value under the control of the reader object. Again, this is deemed inherent because if an empty bit were implemented, it would be meant for reading so that the system knows when the queue is empty.

d) Dretzka, as modified, has further taught notifying the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object and in response to the queue value. When the empty bit is clear and noted by the reader object, the data exists in the queue and should be handled.

e) Dretzka, as modified, has further taught unloading the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. Again, in response to the notification, the data will be moved from buffer 110 to 120-4.

f) Finally, note that while Dretzka hasn't recited objects, as claimed, implementing the above functions via object execution is obvious for the reasons set forth in claim 10.

55. Referring to claim 15, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation), further comprising:

a) a second buffer. See Fig.5, component 120-4.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. See Fig.3 and Fig.5. Data is unloaded from buffer 110 to buffer 120-4 (second buffer) under control of the second transfer object, and then moved from buffer 120-4 to an additional buffer and interface under control of the third object.

c) Note that while the third object is not explicitly recited in Dretzka, implementing the third data transfer via object execution is obvious for the reasons set forth in claim 10.

56. Referring to claim 17, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation) wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.5 and note that, in general, the first object retrieves data from a source and stores it in a next buffer 110. Likewise, the second object retrieves data from a source (buffer 110) and stores it in a next buffer 120-4. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

b) the processor is operable to execute an object factory and to generate the object code under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted, the first and second object codes will be generated and invoked so that data may be transmitted.

57. Referring to claim 18, Dretzka, as modified, has taught the computing machine of claim 10 (under a second interpretation) wherein the processor is further operable to package the generated data into a message that includes a header and the data under the control of the second data-transfer object. See Fig.3 (at least the level 3 object code), and note that the second object begins packaging the data into a message.

58. Referring to claim 45, Dretzka has taught a method comprising:

a) receiving a message that includes data and that includes a message header that indicates a destination address of the data, the destination address corresponding to a software application. See Fig.4 and Fig.6. Note that a message comes in with a 1-byte header (note level 2.5) that will indicate, upon review by the receiving end, a destination of either input list 230-x or message buffer 220-x (column 7, line 25, to column 8, line 31). Clearly, the message must be received by some corresponding application.

b) loading into a first buffer with via first data-transfer, the received data without the message header, the first buffer corresponding to the destination. See Fig.6 and note that the data is loaded into buffer 220-4 based on the 1-byte header. Note that the 1-byte message header is removed prior to storage in 220-4 according to Fig.4.

c) unloading the data from the buffer with via second data-transfer. See Fig.4 and Fig.6 and note that data is ultimately unloaded from buffer 220-4.

d) processing the unloaded data with an application corresponding to the destination. See Fig.4 and note that after the data is unloaded, it will be sent to the processor where inherent processing on that data will commence.

e) Dretzka has not taught executing first and second data-transfer objects. However, object-oriented programming and objects are known and have known advantages in the art. Therefore, particular features of Dretzka may be implemented as objects. For instance, a loader for loading data into a buffer may be implemented as an object. Likewise, an unloader for unloading data from a buffer may be implemented as an object. Since object-oriented programming has known benefits to at least some programmers, such as making programs easier to manage and keep track of, it would have been obvious to one of ordinary skill in the art at the time of the invention to

modify Dretzka such that the processor executes first and second data transfer objects to carry out the respective transfers.

59. Referring to claim 46, Dretzka, as modified, has taught the method of claim 45. Dretzka has not taught that processing the unloaded data comprises processing the unloaded data with a thread of the application corresponding to the destination. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, the system may switch to a second thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor processes the unloaded data with a thread.

60. Referring to claim 47, Dretzka, as modified, has taught the method of claim 45, further comprising:

a) generating a queue value that corresponds to the presence of the data in the buffer. See Fig.4 and Fig.6. The queue object will generates the "more" bit, which corresponds to the presence of data in the buffer. This bit, when set in a certain manner, will allow for the informing of level 4 of a complete message.

b) notifying the second data-transfer object that the data occupies the buffer in response to the queue value. When the "more" bit is set in the appropriate manner and noted by the reader object, the data may be transferred to the next-level buffer by informing (notifying) the next level that data is ready to be transferred. See Fig.4 and Fig.6.

c) wherein unloading the data comprises unloading the data from the buffer with the first data-transfer object in response to the notification. Again, in response to the notification, the data will be moved from level 3 buffer to level 4 buffer. See Fig.4 and Fig.6.

61. Referring to claim 48, Dretzka, as modified, has taught the method of claim 45, further comprising wherein receiving the message comprises receiving the message with the first data-transfer object. See Fig.4 and Fig.6. Some code/object will cause the receiving of the data. That object is the first data transfer object. Note that while the communication object is not explicitly recited in Dretzka, implementing the receiving via object execution is obvious for the reasons set forth in claim 45.

62. Referring to claim 49, Dretzka, as modified, has taught the method of claim 45, further comprising:

a) receiving the message comprises retrieving the message from a bus with a communication object. See Fig.4 and Fig.6. Some code/object will cause the receiving of the data from a bus. That object is the communication object. Note that while the communication object is not explicitly recited in Dretzka, implementing the receiving via object execution is obvious for the reasons set forth in claim 45.

b) transferring the data from the communication object to the first data transfer object. See Fig.4 and Fig.6. Note that after the data is received, it is passed to the first data transfer object which at least stores it in buffer 220-4.

63. Referring to claim 50, Dretzka, as modified, has taught the method of claim 45. Dretzka has not explicitly taught generating the message header and the message with a hardwired pipeline accelerator. However, Official Notice is taken that hardwired pipelined processors and

their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's processor (Fig.1, component 11) such that it includes a hardwired pipeline for accelerating execution. Note that in this series of rejections, processor 11 is the unit which generates the messages according to Fig.3.

**Allowable Subject Matter**

64. Claims 19-24, 51-54, and 62 are allowed. Please review all claims for informalities.

65. The examiner asserts that the term "object" in the allowable (and non-allowable) claims is hereby limited to an object as is known in the art of object-oriented programming. "Object" cannot be interpreted as anything but an object-oriented object, which binds data with methods that operate on that data. See page 27 of applicant's arguments. Variations of "object", e.g., "objects" are limited in the same fashion.

66. The examiner asserts that the term "publish" in the allowable (and non-allowable) claims is hereby limited to the publish action as is known in the art of publish/subscribe messaging protocol(s). "Publish" cannot be interpreted as anything but what is consistent in publish/subscribe communication. Specifically, to publish is to transmit data to one or more destinations from a source where the source does not know the destination or generate the address of the destination. See page 28 of applicant's arguments. Variations of "publish", e.g., "publishing", "published", etc., are limited in the same fashion.

**Response to Arguments**

67. Arguments pertaining to claims no longer rejected under Dretzka are hereby moot and will not be addressed by the examiner.

68. With respect to the argument of the rejection of claims 10 and 37:

a) the examiner asserts that even if a destination address must be generated (to specify one of modules 10, 20, and 30 for communication in Fig.1), this address may still be considered as being separate from the generated data (e.g., the payload) and, therefore, the generated data does not include data destination information. That is, the message data and the address of the component to which the message data will be sent are distinct. Furthermore, please note that the examiner still maintains his previous interpretation as well (regarding the 1 and 3-byte headers).

b) the examiner admits that Dretzka has not taught "objects" as applicant has argued. However, the examiner asserts that it would have been obvious for Dretzka's transfers to occur via object execution due to the well-known advantages of object-oriented programming. See the new ground of rejection above.

c) the examiner admits that Dretzka has not taught "publishing" as applicant has argued. Therefore, the rejection of claim 37 under Dretzka has been withdrawn. However, note the new grounds of rejection under Bass for claim 37.

69. With respect to the argument of the rejection of claim 45:

a) the examiner admits that Dretzka has not taught "objects" as applicant has argued. However, the examiner asserts that it would have been obvious for Dretzka's transfers to

occur via object execution due to the well-known advantages of object-oriented programming. See the new ground of rejection above.

b) the examiner asserts that since it is obvious to modify Dretzka to be in an object-oriented environment, the it follows that the messages in the system must be compatible with an object interface.

**Conclusion**

70. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. Applicant is reminded that in amending in response to a rejection of claims, the patentable novelty must be clearly shown in view of the state of the art disclosed by the

references cited and the objections made. Applicant must also show how the amendments avoid such references and objections. See 37 CFR § 1.111(c).

Oki et al., "The Information Bus - An Architecture for Extensible Distributed Systems", 1993, pp.58-68, discusses the publish/subscribe protocol.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DAVID J. HUISMAN whose telephone number is (571)272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/David J. Huisman/
Primary Examiner, Art Unit 2183